CS 202 - Computer Science II

Project 2

Due date (FIXED): Wednesday, 2/6/2019, 11:59 pm

Objectives: The two main objectives of this project is to test your ability to (1) create and use structs with arrays, (2) work with pointers, pointer arithmetic, pass by-Value, pass by-Reference, pass by-Address, and (3) design, implement and test a solution to a given problem. A review of your knowledge of arrays, iostream, file I/O and C-strings is also included.

Description:
For this project, you are to create a program that will assist users who want to rent a car. You are given a **datafile with 5 different cars file** (the file is a priori known to have exactly 5 entries, each following the same data layout), and you must **read in all** of the **car data** from the **file** and **store** it in an **array of structs**. You must also **create a User Menu** with the functionality defined below. Although an example file is provided (Cars.txt), for grading purposes your project will be tested against a different test file that will not be provided to you beforehand. Our test file will be in the same format as the example file.

The RentalCar struct will contain the following data members:

- **year**, an `int` (year of production)
- **make**, a C-string (`char` array of 10 maximum size)
- **model**, a C-string (`char` array of 10 maximum size)
- **price**, a `float` (price per day)
- **available**, a `bool` (1 = true; 0 = false; try to display true/false using the "`std::boolalpha`" manipulator like: `cout << boolalpha << boolVariable;`)

The User Menu prompt must have the following 7 entries, each implementing a specific functionality when selected:

- **1) Ask** the user for the **input file name**, and then **read ALL** data from that **file**. The data have to be stored into an **array of structs**.
- **2) Print out ALL** data for each of the cars to the **terminal**, preceded by their current **index number** in the array.
- **3) Print out ALL** data for all of the cars to a **separate output file** (when the user chooses menu entry 3, they should also get **asked** for an **output file name**), following the original data format of the provided file (with **no index number**).
- **4) Sort** the cars (i.e. sort the array of structs used to store all data) by **ascending price**.
- **5) Ask** the user for **how many days** they want to rent a car. Then **print to the terminal only the available** cars preceded by their current **index number** in the array, **sorted** by ascending price, as well as the **total estimated cost** to make the rent (number of days multiplied by price per day).
- **6) Ask** the user **which car** they want to rent (expected user input is an **index number** for the array of structs) and for **how many days**. If the corresponding car is **not available**, print a **warning message** to **terminal**. If it is **available**, mark it as rented (modify the available member appropriately) and print out to **terminal** a **success message** that mentions the **total cost** (number of days multiplied by price per day).
- **7) Exit** program.

The following minimum functionality and structure is required:

- Ask the **user** for the **input file** name.
- The list of cars must be stored in an **array of structs**.
- Use **character arrays** to hold your strings (i.e., C-style) exclusively (using the `string` data type is still not allowed).
- Write **multiple functions** (Hint: each menu option should be a function).
- You are free to use **pass by-Value, pass by-Reference**, **pass by-Address** for your function parameters. (*Note*: Remember that using pass by-Value will make the function work on a local internal copy of whatever variable you pass as an argument, therefore the change will not be made on the actual argument itself, and it will be left unaffected after the function call is complete).
- Write your **own C-string length**, **compare**, **copy**, **concatenate** functions. These are required, even if they are not used within the rest of your program. Their prototypes will have the form (use the prototypes exactly as provided, with **`char *`** parameters):

```
// counts characters in str array until a NULL-character '\0' is
found, then it returns that number excluding the '\0' one
// the return type size_t represents an unsigned integral number
large enough to contain the maximum possible number of a storage
size that can appear on a target architecture
size_t myStringLength(const char * str);

// returns 0 when the C-strings match, i.e. their characters are
equal one-by-one until a NULL-character '\0' is found in both
strings and at the same position as well
// returns a value <= -1 if the first character that does not
match has a lower value in str1 than in str2
// returns a value >= 1 if the first character that does not
match has a higher value in str1 than in str2
int myStringCompare(const char * str1, const char * str2);

// copies characters from source to destination array until a
NULL-character '\0' is found in source, then it NULL-terminates
destination too
// returns a pointer to the destination array
char * myStringCopy(char * destination, const char * source);

// appends the content of source to the destination array
// this means that the NULL-terminator of destination is
overwritten by the first character of source and a NULL-character
'\0' is appended at the end of the concatenated Cstring in
destination
// returns a pointer to the destination array
char * myStringCat(char * destination, const char * source);
```

- The other functionality and structure of the program should remain the **same as Project #1**, including **writing to screen** and **file** and **restrictions on string** libraries, **global variables** and **constants**, etc.

Sample Output for menu option 2 (before sorting the array by ascending price):

[0]: 2014 Toyota Tacoma , $115.12 per day , Available: false

[1]: 2015 Ford Fusion , $90.89 per day , Available: true

[2]: 2009 Dodge Neon , $45.25 per day , Available: false

[3]: 2015 Ford F150 , $112.83 per day , Available: true

[4]: 2016 Subaru Outback , $71.27 per day , Available: true

Sample Output for menu option 5 (for 2 days of renting, only showing available ones, and with the array already sorted by ascending price, indexes correspond to the sorted array entries):

[1]: 2016 Subaru Outback , Total Cost: $142.54

[2]: 2015 Ford Fusion , Total Cost: $181.78

[3]: 2015 Ford F150 , Total Cost: $225.66

The completed project should have the following properties:

- Written, compiled and tested using Linux.
- It must compile successfully using the g++ compiler on department machines.
  Instructions how to remotely connect to department machines are included in the Projects folder in WebCampus.
- The code must be commented and indented properly.
  Header comments are required on all files and recommended for the rest of the program.
  Descriptions of functions commented properly.
- A one page (minimum) typed sheet documenting your code. This should include the overall purpose of the program, your design, problems (if any), and any changes you would make given more time.

Turn in: Compressed .cpp file and project documentation.

Submission Instructions:

- You will submit your work via WebCampus
- Name your code file proj2.cpp
- If you have header file, name it proj2.h
- Compress your:
  1. Source code
  2. Documentation
  Do not include executable
- Name the compressed folder:
  PA#_Lastname_Firstname.zip
  ([PA] stands for [ProjectAssignment], [#] is the Project number)
  Ex: PA2_Smith_John.zip

Verify: After you upload your .zip file, re-download it from WebCampus. Extract it, compile it and verify that it compiles and runs on the NoMachine virtual machines or directly on the ECC systems.

- Code that does not compile will be heavily penalized –may even cost you your *entire* grade–. Executables that do not work 100% will receive partial grade points.
- It is better to hand in code that compiles and performs partial functionality, rather than broken code. You may use your Documentation file to mention what you could not get to work exactly as you wanted in the given timeframe of the Project.

Late Submission:

A project submission is "late" if any of the submitted files are time-stamped after the due date and time. Projects will be accepted up to 24 hours late, with 20% penalty.